Leveraging Pattern Semantics for Extracting Entities in Enterprises

Fangbo Tao[†], Bo Zhao[‡], Ariel Fuxman[♭], Yang Li[♮], Jiawei Han[†]
[*†]ftao2@illinois.edu, [‡]bozhao@linkedin.com, [♮]yangli@cs.ucsb.edu, [†]hanj@illinois.edu

[†]University of Illinois
Urbana, IL, USA

[‡]LinkedIn
Mountain View, CA, USA

[♭]Microsoft
Mountain View, CA, USA

[♮]UC Santa Barbara
Santa Barbara, CA, USA

ABSTRACT

Entity Extraction is a process of identifying meaningful entities from text documents. In enterprises, extracting entities improves enterprise efficiency by facilitating numerous applications, including search, recommendation, etc. However, the problem is particularly challenging on enterprise domains due to several reasons. First, the lack of redundancy of enterprise entities makes previous web-based systems like NELL and OpenIE not effective, since using only high-precision/low-recall patterns like those systems would miss the majority of sparse enterprise entities, while using more low-precision patterns in sparse setting also introduces noise drastically. Second, semantic drift is common in enterprises ("Blue" refers to "Windows Blue"), such that public signals from the web cannot be directly applied on entities. Moreover, many internal entities never appear on the web. Sparse internal signals are the only source for discovering them. To address these challenges, we propose an end-to-end framework for extracting entities in enterprises, taking the input of enterprise corpus and limited seeds to generate a high-quality entity collection as output. We introduce the novel concept of Semantic Pattern Graph to leverage public signals to understand the underlying semantics of lexical patterns, reinforce pattern evaluation using mined semantics, and yield more accurate and complete entities. Experiments on Microsoft enterprise data show the effectiveness of our approach.

1. INTRODUCTION AND MOTIVATION

Access to an organized information network or knowledge base is critical for many real-world applications. Most real-world information is unstructured, interconnected, noisy, and often expressed in the form of text. This inspires constructing an organized, semi-structured knowledge base from the large volume of noisy text data. Such formal and structural representation of information has the advantage of being easy to manage and reason with, which can greatly facilitate many artificial intelligence applications, such as semantic search, reasoning and question answering. To achieve this goal, knowledge bases such as DBpedia [2] and Freebase [3] were manually constructed. However, due to the laborious, time consuming, and costly extracting and labeling process, these knowledge bases are often restricted by a very limited coverage. Recently, automatically constructed knowledge bases including YAGO [22], NELL [5] and Reverb [12] have emerged. However, it is still challenging to make such systems with both satisfactory coverage and quality. How to automatically construct a high-quality knowledge base from large amount of unstructured and noisy text data remains an open research problem.

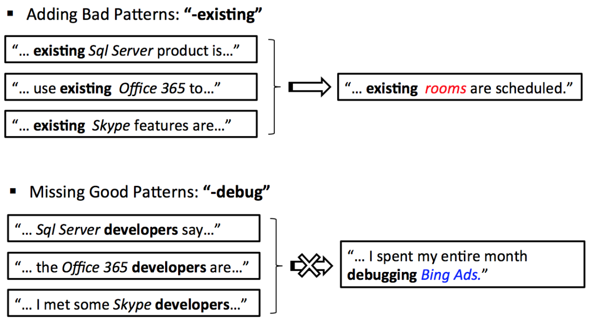

Adding Bad Patterns: "-existing"
"... existing Sql Server product is..."
"... use existing Office 365 to..."
"... existing Skype features are..."
"... existing rooms are scheduled."
Missing Good Patterns: "-debug"
"... Sql Server developers say..."
"... the Office 365 developers are..."
"... I met some Skype developers..."
"... I spent my entire month debugging Bing Ads."

Figure 1: Two examples showing how sparsity makes entity extraction challenging in enterprises.)

Most previous studies on automatic knowledge base construction focus on open domain. Namely, they extract information from web-scale corpus and try to cover as many entities as possible. Despite the fact that these open domain efforts cover millions of entities, most lesser known or domain specific entities are not captured by them. Therefore, in closed domains (e.g., enterprises, governments, etc.), in order to better facilitate the knowledge-inspired applications, we have to build domain-specific knowledge bases. In this work, we focus on one particularly important domain, enterprises, and study the *Enterprise Entity Extraction* problem. Our goal is to harvest a comprehensive entity set from the enterprise corpus. The set should cover important entities within enterprises (e.g., products, teams, techniques, etc.) and provide categorical information for these entities. This can be seen as the first step towards building a complete enterprise knowledge base, which should eventually contain attributes, relations and brief summaries of these entities.

[*]This work was done while the first four authors were employed at Microsoft. Fuxman's current affiliation: Google.

The *Enterprise Entity Extraction* task is very related to the Web Information Extraction [9, 5, 14, 21] problem, which aims to extract interesting entities (and potentially the relations among them) from Web corpus. Unfortunately, the techniques proposed for Web entity extraction can hardly be applied to enterprises. This is because enterprise corpora exhibit very different characteristics compared with the Web corpus, which leads to the following unique challenges for extracting entities in enterprises.

1. **Data Sparsity:** Enterprise entities exhibit much less redundancy than public entities in the Web. Therefore, the extracting techniques based on frequencies or statistics can hardly be utilized in enterprises.

2. **Semantic Drift:** Many entities in enterprises have special meanings different from the popular meanings in external world. For example, within Microsoft, "Blue" could mean "Microsoft Blue" instead of the color. Even for the same entities, the internal usage could be very different from the external usage. For example, for the entity "Windows", external usage mainly focuses how to install and use it, while internal usage cares more about implementation and design. Therefore, it is inappropriate to directly explore distributional semantics [17] for iteratively extracting enterprise entities.

3. **Low Public Coverage:** Most internal entities are not covered by public knowledge bases. Some are even not covered by the Web. Therefore, it is very hard to leverage public resources (e.g., Wikipedia, Google) to provide complementary information. For instance, the entity "Cloud ML" can only be found in Microsoft.

Data sparsity suggests to consider public data to alleviate deficient redundancy. However, due to Challenges 2 & 3, incorporating public signals on entities directly introduces much noise via Semantic Drift and also, not feasible for internal entities because of Low Public Coverage. Thus, instead of harvesting entity signals from public data, pattern-level approach is considered to tackle these challenges.

Most previous studies [9, 5, 14, 21] utilize general high precision/low recall patterns (e.g. Hearst Patterns) to extract entities. Namely, they extract lexical patterns, optionally with some fixed form (Hearst Patterns), from the contexts of seed entities, evaluate the patterns' precision, and promote only a few top ones for further entity extraction. While this is a good fit for the redundant Web corpus, it causes severe coverage issues in enterprise entity extraction since the enterprise corpus is very sparse. Based on our experiment on Microsoft corpus, under-utilizing these accurate patterns leads to less than 50% of enterprise entities being successfully extracted. That is to say, most entities in enterprise documents have contextual patterns with deficient mentions to be justified. But these patterns have highly relevant semantics to target entities, which we define as **Sparse Patterns**. Therefore, to better cover the remaining entities, some additional patterns (likely sparse) need to be incorporated.

Harvesting good patterns is very challenging, since the data sparsity issue makes it difficult to judge the quality of the patterns. As a result, the following two types of errors will emerge:

1. **Adding Bad Patterns:** Some bad patterns (e.g., very general patterns) may be wrongly judged as good ones. For example, as shown in Figure 1, pattern **"existing X"** is general enough for all kinds of entities, but it is mistakenly evaluated as a good pattern to distinguish "projects", due to the fact that **"existing X"** appears with all three positive seeds. Therefore, noun **"room"** would be implied as a "project" by the line "...*existing* **rooms** are scheduled...". Patterns like **"existing X"** are general enough to appear with any nouns. Using limited enterprise documents, we may not have enough signal to filter them out.

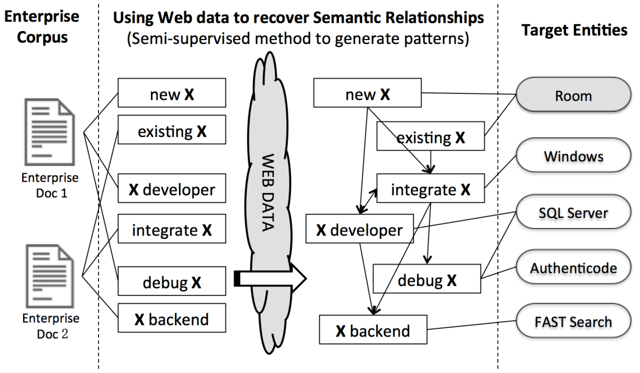


Figure 2: A toy example showing how we integrate enterprise corpus with web data to construct semantic pattern graph and improve entity quality.

2. **Missing Good Patterns:** Some good patterns (especially sparse patterns) may be erroneously identified as bad. For instance, as shown in Figure 1, we already observe that **"X developer"** is a solid pattern since it appears with all the positive seeds. However, lexical pattern **"debug X"** was skipped by the extractor, since it never appears with seed entities. Therefore, the good pattern **"debug X"** is totally lost without any evidence to prove its efficacy. With enough background knowledge, any human expert can conclude that **"debug X"** has similar semantic to **"X developer"**. Moreover, an asymmetric relation holds here that **"X developer"** is more general than **"debug X"**. With high probability, an entity "X" that can be used as "debug X" can also be used as "X developer"; the other direction doesn't hold. This asymmetric relation implies that if **"X developer"** is a strong pattern to extract "projects", any noun "X" mentioned as **"debug X"** in the corpus should also be extracted as "projects" because the asymmetric relation makes the "X" possible to be used as **"X developer"**. However, using only enterprise corpus cannot discover the semantic relation between these two patterns, which leads to poor performance of previous entity extractors.

To address these two problems, we developed a semi-supervised pattern extraction method and leverage Web data to understand the pattern semantics. As illustrated in Figure 2, semi-supervised pattern extraction can help extract more lexical patterns from enterprise text, while Web text can be used to build semantic relations between extracted patterns, which can be further utilized to induce sparse yet effective patterns. The built Semantic Graph helps to identify bad general patterns and good sparse patterns mentioned above, thus fixing the sparsity issue.

In this paper, we tackle the enterprise entity extraction problem via bootstrapping. We take a few seed entities plus an unlabeled enterprise corpus as input and produce a high-quality entity set as output. With insufficient knowledge of lexical patterns in enterprise corpus, we leverage public signals (i.e., Web text) to understand the underlying semantic relations among sparse lexical patterns. For this purpose, we propose a novel concept called Semantic Pattern Graph (SPG), which is a hierarchical graph structure describing the relations between lexical patterns. To the best of our knowledge, this is the first work that models semantic relations between lexical patterns for entity extraction.

Our contribution can be summarized as follows: (1) We develop an end-to-end framework for extracting entity in enterprises. (2) We propose the novel concept of Semantic Pattern Graph to utilize semantic relations between patterns for entity extraction. (3) We conduct comprehensive experiments on Microsoft enterprise corpus to justify the effectiveness of our system.

2. RELATED WORK

Recently, several open information extraction systems have been created. Most of them focus on entity and relation extraction on web-scale data. Knowitall [9, 10, 8] combined pattern-based and list-based extraction to achieve recall improvement. They used a set of generic, domain independent extraction patterns (mostly Hearst patterns [16]) to extend a set of seed concepts. NELL [5] is another open information extractor for harvesting entities and relations from the web. It used Coupled Pattern Learner [6], which extracts lexical patterns with part-of-speech (POS) restrictions from positively labeled data, to identify new entities. A restrict filtering constraint is applied to guarantee only high-precision/low-recall patterns are promoted. SPIED [14] used a similar way of pattern generation but scored patterns using both labeled and unlabeled entities. These three systems were designed to work with redundant entity mentions (e.g., web data) and performed poorly on sparse enterprise settings. In our tests of these pattern generators, we observe that high-precision/low-recall patterns they used can hardly cover 50% of our target entity set. We provide labeled data for SPIED and both precision and recall are quite unsatisfactory. Other open information extraction systems like ReVerb [12, 11] and OLLIE [21] used verbal patterns or some extraction templates, which may fail to work well in sparse environment. Moreover, Poon and Domingos [18] showed that open information extraction systems extracted low accuracy relational triples on a small corpus. Hence, two things are required in enterprise entity extraction: 1). More low-precision patterns need to be leveraged to cover more entities. 2). Open signals from the web need to be incorporated to improve accuracy of those sparse patterns.

Our approach employs a semi-supervised bootstrap learning method, which begins with a small labeled set of target entities, trains a learning/ranking model, and uses that model to label more data and so on. Yarowsky [27] applied bootstrapped learning on word sense disambiguation. Later Riloff [19] used a set of seed entities to learn rules for entity extraction from unlabeled data and extended it to multi-class learning in [24]. Similar methods are also used in many set-expansion works. SEAL [25, 26] is a web-based set expansion system that uses wrappers (i.e., page-specific extraction rules) to extract more entities. It takes advantage of both page structure and text from webpages. For many cases where only text data is available, several IE systems [14, 1, 4, 20] also applied bootstrap method, specify a small set of domain-specific seed instances as input, then alternately learn patterns from seeds, and extend seeds from patterns. Our system inherits the semi-supervised bootstrap learning method for enterprise entity extraction.

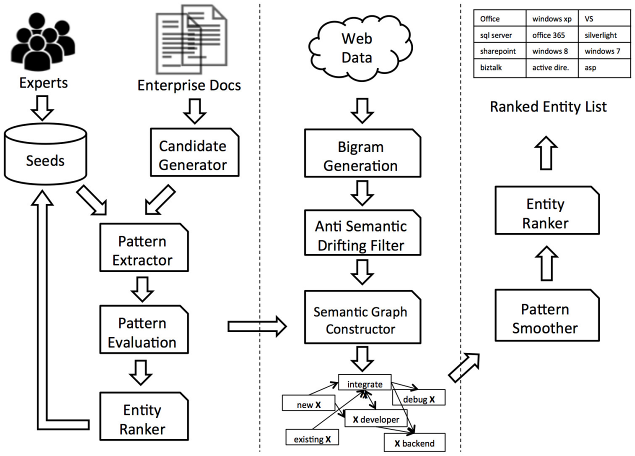


Figure 3: System Infrastructure

A distinctive feature of our system is its use of Semantic Pattern Graph, which is derived from web-scale data, to rescore patterns and ease the sparsity issue. The pioneering work for pattern generation and scoring by Hearst [16] manually evaluated generated patterns to extract hypernym-hyponym pairs. Previous systems [13, 7, 23] used fully labeled corpus to score rules. Later, Carlson et al. [6] assessed patterns by precision and only promote patterns with high precision. We implemented their pattern assessment method in this paper for comparison and shown that his strategy works poorly on small corpus due to the coverage of high-precision patterns are very small. Gupta and Manning [15, 14] predicted labels of unlabeled entities to score patterns using features like distributional similarity and edit distances. None of the above works well enough in a enterprise setting due to the sparse and biased signals of patterns. Our system outperforms them by utilizing outside signals from the web to adjust the biased evaluation of patterns, which is computed merely using mentions in enterprise corpus.

3. APPROACH

We apply a set expansion framework to extract enterprise entities class by class. The reason is that letting domain experts provide a seed set for each entity class is relatively cheap. Moreover, in some enterprises (e.g., Microsoft), a partial taxonomy has been built by their employees, which can serve as the seed set directly. For ease of exposition, we present the approach below for extracting entities for one class C. It can easily be generalized to multiple classes. The bootstrapping process involves following steps (also shown in Figure 3).

1. Generating candidate pool and labeling data: By scanning all the documents in the enterprise corpus, a candidate pool containing all the possible enterprise entities is generated. Positive entity seeds from class C is

also provided by enterprise insiders/experts. Negative seeds are automatically generated from the candidate pool using heuristics, i.e., capitalized ratio;

2. Pattern Extractor: Contextual patterns are created using context text around all the candidate entities in both seed set and candidate pool.
3. Evaluating patterns using Multinomial Naive Bayes model.
4. Ranking entities and adding top confident ones into seeds. A semi-supervised framework is used to extract more low-precision/sparse patterns. Each iteration adds the top positive/negative entities into the seed set and re-scores patterns, after certain amount of iterations.
5. Semantic Pattern Graph Construction: Bi-grams are first extracted from web text for semantic recovery. An Anti Semantic Drifting Filter is then applied to avoid inconsistency of single pattern's semantic in different domains. And with the extended pattern list generated in enterprise corpus, a Semantic Graph is built using bi-gram data and enterprise patterns. Details will be discussed in Section 4.
6. Smoothing pattern scores: A smoother is applied to re-evaluate each pattern based on their original score and graph structure, details will be discussed in Section 4.
7. Ranking entities: Smoothed patterns for the class are applied to the entity candidates. A Multinomial Naive Bayes classifier ranks the candidate entities and adds the top entities to C's dictionary.

In this section, we will explain steps 1-4 in detail. Steps 5-7 will be covered in the next section.

3.1 Generation of Candidate Pool

Different from the previous Web-scale entity extractors, we generate all possible entity candidates as our first step. Many enterprise entities and their patterns might be very sparse, such that using only seed patterns may never discover them. Finding these sparse entities first takes their sparse patterns into account in the process. Two restrictions are applied in the generation of candidate pool. First, the target term has a part-of-speech (POS) restriction, which is the POS tag sequence of the candidate phrase. Second, a noun phrase will be considered as a candidate only if the phrase appears at least once in the corpus as capitalized form.

In our framework, we automatically generate negative seeds from the candidate pool. For a candidate term t, sets C_t and U_t denote the capitalized and uncapitalized mention sets in the corpus. We require $\frac{|C_t|}{|C_t|+|U_t|} \leq threshold$ to consider t as a negative seed. We set $threshold = 0.1$ in our experiments.

3.2 Generation of Patterns

We use lexico-syntactic surface word patterns to extract entities from candidate pool. They are created using contexts of words or their lemmatized form within a window of one before or after a entity candidate in the candidate pool. Here we collect as many patterns as possible, even include those that never appear with any positive/negative samples. These "invisible" patterns might be quite useful after understanding its semantic relations in the smoothing phase. Therefore, we take them into account in every step. We generate flexible pattern by removing {"a", "an", "the"} when matching patterns to the text.

Two reasons are considered to pick one as the window size instead of two or more: 1) Bigger window size may introduce more noisy patterns that are irrelevant to the entity. 2) It is more interpretable to construct semantic pattern graph on unigram patterns.

To simplify the notation, we will use "+" or "−" to indicate the relative location of lexical patterns. "+" means the pattern appears after the entity and "−" means the pattern appears before the pattern. Thus, pattern **"existing X"** would be **"−existing"** and **"X developer"** would be **"+developer"** and so on.

3.3 Scoring Patterns using Semi-supervised NB

A Semi-supervised Naive-Bayes (**Semi-NB**) model is applied here as the ranker and classifier for our task. We treat lexical patterns as features and treat every candidate in the candidate pool as testing data. User-provided positive seed set and automatically generated negative seed set are training data in our Semi-NB setting. For each target entity e in both training and testing data, a feature vector is generated, each feature score on feature f is calculated as:

$$S(e, f) = \log\left(count(e, f) + 1\right)$$

The $count(e, f)$ here means the total mention count of target entity e with feature (pattern) f. With the real-valued feature scores, we use a multinomial Naive Bayes to calculate pattern scores and entity score. Entity scores $T(e)$ are calculated as follows:

$$\begin{aligned} T(e) = \log\frac{P(+|e)}{P(-|e)} &= \log\frac{P(+)\prod_f P(f|+)^{S(e,f)}}{P(-)\prod_f P(f|-)^{S(e,f)}} \\ &= \sum_f S(e,f)(\log P(f|+) - \log P(f|-)) + \log\frac{P(+)}{P(-)} \end{aligned}$$

Where $P(f|+) = \frac{count(f,+)+1}{count(+)+|F|}$, $P(f|-) = \frac{count(f,-)+1}{count(-)+|F|}$. F is the set of all patterns. To evaluate each pattern, we also define normalized pattern score $R(f)$ as

$$R(f) = \lambda(\log P(f|+) - \log P(f|-))$$

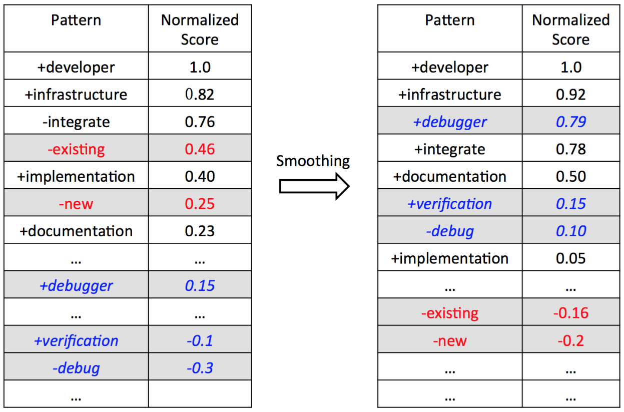
Where λ is the normalization constant to normalize $R(f)$ to [-1, 1]. The value of λ is determined by the range of $\log P(f|+) - \log P(f|-)$ across all features. Positive $R(f)$ shows that pattern f is good signal for extracting enterprise entities, and vice versa. Therefore we rewrite $T(e)$ as:

$$T(e) = \sum_f S(e,f)R(f) + c$$

Moreover, to ease sparsity in this phase, we applied iterative semi-supervised mechanism into the process. As shown in Algorithm 1 and Figure 3, we iteratively add top positive candidates and top negative candidates into the seed set. We re-calculate pattern scores after certain amount of iterations (here we chose 10 as our default iteration rounds) and use the new pattern scores to rank entities. Experimental results in Section 5 show that the semi-supervised mechanism works well for alleviating sparsity by adding more seeds and discover more patterns afterwards. M is chosen as 20 in our experiment.

Algorithm 1: Semi-NB for Enterprise Entity Extraction

```
Data: Enterprise Corpus
Result:
Initialization;
for $i = 0, i < iteration\#, i{+}{+}$ do
    Calculate pattern score using seed set;
    Rank entities in candidate pool;
    Add top M positive entities into the pos seed set;
    Add top M negative entities into the neg seed set;
Calculate pattern score using updated seed set;
Smooth patterns using SPG;
Rank entities in candidate pool;
```



Pattern	Normalized Score
+developer	1.0
+infrastructure	0.82
-integrate	0.76
-existing	**0.46**
+implementation	0.40
-new	**0.25**
+documentation	0.23
…	…
+debugger	*0.15*
…	…
+verification	*-0.1*
-debug	*-0.3*
…	

Pattern	Normalized Score
+developer	1.0
+infrastructure	0.92
+debugger	*0.79*
+integrate	0.78
+documentation	0.50
+verification	*0.15*
-debug	*0.10*
+implementation	0.05
…	…
-existing	**-0.16**
-new	**-0.2**
…	…
…	…

Figure 4: Contrast of scores before and after smoothing. Red/bold patterns are the mistakenly highly ranked general patterns, and blue/italic patterns are “good” sparse patterns.

3.4 Learning Enterprise Entities

After the smoothing phase (will be discussed in Section 4), we applied the learned patterns to the extracted entity candidates using the multinomial naive-bayes classifier. The candidates are ranked to indicate how possible it is as an enterprise entity. We then consider the top positive ones as our extracted entities and output them.

4. SEMANTIC PATTERN GRAPH

As discussed in Introduction, due to the lack of redundancy of entity mentions compared with web-scale data, enterprise entity extraction is especially challenging. To be more concrete, we show here the pattern scores we get without leveraging any public resources or smoothing in Figure 4. All the scores are normalized to scale [-1, 1]. On the left side, we show some selected pattern scores in the ranked list before leveraging Semantic Pattern Graph as a smoother. The numbers on the second column denote how much degree a pattern would tell a “good” enterprise entity. Positive number means the pattern is a positive signal for enterprise entity and vice versa. Clearly we can see three different types of patterns in Figure 4.

1. **Type I - Non-Sparse Good Patterns**: patterns like *“+develop”*, *“+infrastructure”* and *“−integrate”* belong to this category. These patterns are good indicators for chosen Enterprise Entity class C and usually rank pretty high in the pattern list. They appears frequently with positive seeds and infrequently with negative seeds, and they normally have rich semantics strongly correlated to class C;
2. **Type II - Bad General Patterns**: patterns like *“−existing”* and *“−new”* belong to this category. These patterns are bad indicators for C since they are too general. They usually do not have any semantics specifically correlated to class C. They happened to appear with most positive seeds and not too many negative seeds, which leads to their high ranks in the pattern list. Normally if the positive seeds that user provided are much more popular than negative seeds (which is commonly the case), bad general patterns will contaminate the pattern list severely. The problem of these patterns being ranked high is that they may discover non-enterprise entities and make our entity set less accurate.
3. **Type III - Sparse Good Patterns**: patterns like *“+debugger”*, *“+verification”* and *“−debug”* belong to this category. These patterns have semantics strongly correlated to class C but they are not as commonly used as Type I patterns. They are good indicators for extracting enterprise entities in class C. However, different from Type I patterns, they stand for more specific/less common operators for class C. In our example, every project can both be “debugged” and be “developed”, but *“−develop”* is much more common. Therefore, Type III patterns usually can not rank very high and they may even possibly be on the negative side. Here, *“−debug”* has score -0.3 because it never appears with the pos seeds and appears with some neg seeds by accident.

Therefore, if we can demote Type II patterns and promote Type III patterns properly in the ranked pattern list, we can ease the trouble that sparsity brings and expect better quality of enterprise entities. To do so, a deeper understanding of the semantic relations between patterns is required. However, it cannot be obtained from enterprise corpus because of Sparsity again! The solution lies outside of the corpus: Web Data. It is possible to recover semantic structure from the web due to the fact that the patterns both enterprise entities and public entities use are shared. Here we define the key concept **Semantic Pattern Graph** as follows:

Definition 1. **Semantic Pattern Graph (SPG)**: A Semantic Pattern Graph is defined as a complete directed graph $G = (V, E)$, in which V is the pattern set and E contains edges representing relations between patterns. In **SPG**, every pair of distinct patterns is connected by a pair of edges.

A sample **SPG** is shown in Figure 5. The scores on the edges are defined as follows:

$$S_{p\to q} = \frac{|E_p \cap E_q|}{|E_p|}, \quad S_{q\to p} = \frac{|E_p \cap E_q|}{|E_q|}$$

Here p, q are two distinct patterns. E_p, E_q denote the entity set that can appear with pattern p, q respectively in web data. If $S_{p\to q}$ is high, meaning that E_q is mostly covered by E_p, in other words, most entities that can appear with p, can also appear with q. Similarly, small $S_{p\to q}$ means that most entities that can appear with p, cannot appear with q. With such graph built, we can roughly divide semantic relations between patterns into three categories.

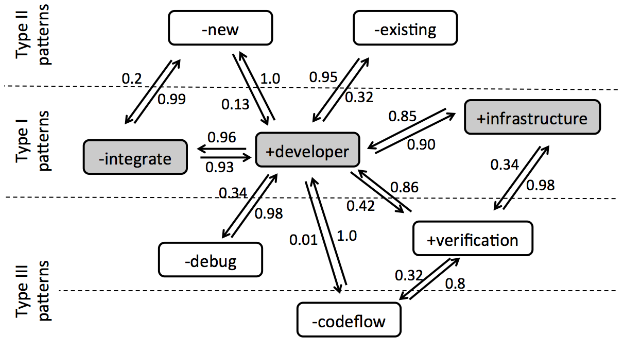


Figure 5: Toy Example of Semantic Pattern Graph from Microsoft dataset. For ease of exposition, only the edges between "+developer", "−integrate", "+infrastructure" and other patterns are shown in this example.

1. **Belonging:** If $S_{p\to q} \ll S_{q\to p}$, we say p is a general pattern of q. Semantically, q belongs to p. Examples are <"−new", "+developer"> and <"+developer", "−debug">, where "+developer" belongs to "−new" and "−debug" belongs to "developer".
2. **Equal:** If $S_{p\to q} \approx S_{q\to p}$ and both of them are large, we say p, q are semantically equal since E_p, E_q share most entities. Examples includes <"+developer", "−integrate"> and <"+developer", "+infrastructure">
3. **Independent:** If $S_{p\to q} \approx S_{q\to p}$ and both of them are small, we say p, q are semantically independent, because E_p and E_q have very little overlap.

Revisit Figure 4 and Figure 5, we observe that Type I pattern are usually in the central layer of SPG, Type II patterns are roughly in top layer and Type III patterns are in the bottom layer. The layers in Semantic Pattern Graph are defined by how general the patterns are. Thus, lower layer entities usually form a **Belonging** relation to higher layer entities, while entities in the same layer normally form **Equal** or **Independent** relations. In other words, Type I patterns **belong** to Type II patterns and Type III patterns **belong** to Type I patterns. Another observation from Figure 4 is that Type I patterns are usually top patterns in the ranked pattern list before smoothing, since they obtains sufficient positive signals from enterprise corpus. These observations inspire our design of smoothing algorithm.

4.1 SPG Construction

In this section, we explain how we construct the **SPG** like Figure 5 from web data. We use Microsoft Web N-gram data, especially bi-gram sub-portion to construct **SPG**. Web Bi-gram dataset contains all possible combinations of patterns and their unigram entities on the web. To avoid unnecessary computation, we construct **SPG** solely based on the patterns appeared in our target corpus. Since web bi-gram data contains much noise, PMI between patterns and entities are used on bi-gram frequency to filter out insignificant bi-grams using a threshold. As an example, PMI of bi-gram "**P E**" will be calculated as:

$$pmi(\mathbf{P}, \mathbf{E}) = \log \frac{p(\mathbf{P}, \mathbf{E})}{p(\mathbf{P})p(\mathbf{E})} = \log \frac{count(\mathbf{PE})}{\sum count(\mathbf{P}*) \sum count(*\mathbf{E})}$$

Here **P** refers to the pattern and **E** refers to the entity. $count(\mathbf{PE})$ denotes the count of bi-gram "**P E**" on the web. $\sum count(\mathbf{P}*)$ denotes the total counts of all bi-grams including pattern **P** and $\sum count(*\mathbf{E})$ is defined similarly.

Anti Semantic Drifting Filter: Another problem we need to address is Semantic Drift of the same pattern on different domains. For example, **"−express"** may have specific meaning (genetically express the gene) in biology domain, while have no such semantics in any other field. That is to say, if we use the whole bi-gram data for constructing SPG for a specific domain, the semantic relations may be contaminated by semantics from other fields. As an example, if we see **"−express"** in computer science corpus, we need to avoid linking this one to **"−translate"**, a pattern that has semantic drift as well in biology domain. In this regard, we apply a filter on web "pattern-entity" pairs to preserve semantics of the target domain as much as possible. We use the entity candidates generated in enterprise corpus only to filter the "pattern-entity" pairs for building SPG. Thus in the **"−express"** case, semantics from biology domain will be filtered out while constructing SPGs for other fields.

After we filter out all the bi-grams containing those patterns and entities. We then calculate score E_p for each pattern p and construct the complete directed graph based on the statistics.

4.2 Smoothing Algorithm

In this section, we introduce the algorithm of using SPG for pattern re-evaluation. Back to our "projects" extraction task for Microsoft internal corpus. We observe in Figure 5 that Type I patterns like **"+developer"** and **"−integrate"**, are in the central layer of a the constructed SPG. Also, we observe that type II patterns, like **"−existing"** and **"−new"**, are more general than Type I patterns in the graph, indicating their **Belonging** relations. Type I patterns are more general than Type III patterns, like **"−debug"** and **"+verification"** in the graph, which also indicates their **Belonging** relations. Based on those observation, a possible solution is to locate the Type I pattern in the graph, and demote their general patterns (Type II) and promote their specific patterns (Type III). Since we also know that Type I patterns, with high possibility, will rank top in the pattern list, we then take the top-K patterns in the ranked pattern list, locate them in the **SPG** and smooth the rest of the graph using an induced graph from SPG. The induced graph is defined as follows:

Definition 2. **Induced Smoothing Graph with Seeds (ISGwS)**: An Induced Smoothing Graph with Seeds is defined as a directed graph $G = (V, S, E)$, in which V is the pattern set, S is the seed set and E contains weighted edges representing smoothing score between patterns in S to patterns in V. Seed set S consists of top-K patterns in the ranked list before smoothing.

As an example, Figure 6 shows the **ISGwS** generated from Figure 5. The new graph contains three seeds **"+developer"**, **"−integrate"** and **"+infrastructure"**. We evaluate how strong the score of a pattern should be smoothed by top-K patterns using formula as follows:

$$T_{p\to q} = \kappa \cdot (S_{q\to p} - S_{p\to q}) \cdot S_{p\to q} \cdot S_{q\to p}$$

Here p is the top pattern, q can be any other patterns in the graph. $\kappa = 4$ is the normalization constant to scale $T_{p\to q}$ to $[-1, 1]$. First term $(S_{q\to p} - S_{p\to q})$ in the formula determine the sign of $T_{p\to q}$. Moreover, smoothing effect

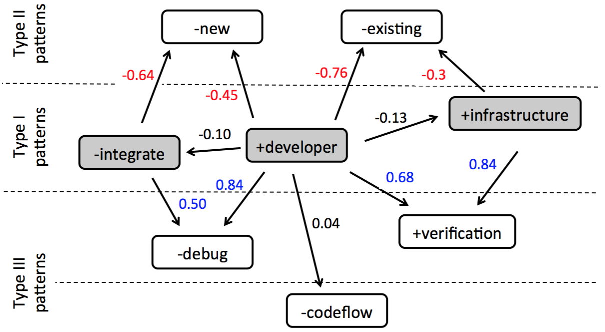


Figure 6: Toy Example of Induced Smoothing Graph with Seeds after scoring.

grows as the difference between $S_{q\to p}$ and $S_{p\to q}$ grows. The second and third terms $S_{p\to q}, S_{q\to p}$ convey two things. 1). The bigger direction $S_{p\to q}$ (or $S_{q\to p}$, depends on demoting or promoting) should be close to 1 to strongly show that there is a clear **Belonging** relation going on. For example, $S_{+developer\to -new}$ should be almost 1. 2). The smaller direction $S_{q\to p}$ (or $S_{p\to q}$) should also be big enough, such that the **Belonging** relation is not because of noise. For instance, pattern **"+codeflow"** in Figure 5 should not be promoted as much as **"−debug"** because that $S_{+developer\to +codeflow}$ is too small.

We get an Induced Smoothing Graph with Seeds by scoring the edges in Figure 5. The new graph is shown in Figure 6. By giving three top patterns, we strongly demote pattern "−new" and "−existing", promote pattern "−debug" and "+verification" and not do much to pattern "−integrate", "+infrastructure" and "+codeflow". The scoring approach satisfies our smoothing goal.

The last step is to average over all top-K patterns in the graph to smooth every other pattern, as follows:

$$R'(q) = (1-\alpha)R(q) + \frac{\alpha}{K} \cdot \sum_{p \in top-K} T_{p\to q} \cdot R(p)$$

Here $R(q)$ and $R'(q)$ denote the pattern score before and after smoothing respectively. We use a linear combination of the original score and smoothed score to represent new score of pattern q. Moreover, we define the following form as *Smoothing Factor*:

$$SF(q) = \frac{1}{K} \sum_{p \in top-K} T_{p\to q} \cdot R(p)$$

After smoothing, we convert the pattern list in left side of Figure 4 to the right side list, as expected. A key problem here is how to choose α and K. α controls the balance between enterprise signals and public signals. K controls the effective size of trustworthy patterns using only local signals. Thorough experiment is conducted in Section 5 to answer this question.

5. EXPERIMENT

In this section, we evaluate the effectiveness of our framework for Enterprise Entity Extraction on real dataset from Microsoft. We will (1) Demonstrate the quality of discovered enterprise entities using several measurement capturing both precision and recall. (2) study the two key parameters α and K in Semantic Pattern Graph smoothing step and understand the philosophy of picking them properly. (3) study the entity coverage of using different pattern generation strategies, to show why it is necessary to consider much more patterns in enterprises than previous web-scale information extractors. (4) case study on 5 specific patterns of Type II and Type III to show the correctness of our analysis in Section 4. All the experiments, if not specifically mentioned, are conducted on a PC with 2.4GHz CPU and 16GB RAM.

5.1 Experimental Setup

We evaluate our system on extracting projects/products entities in Microsoft Internal Answer Forum corpus. The forum has 484,221 discussion threads by Microsoft employees concerning all kinds of technical problems and solutions from the years 2011 to 2014. For each thread, we concat the question and its answers together, and only keep the content for further analysis. We asked Microsoft employees to generate a complete list of projects/products with the help of Microsoft Internal Taxonomy. The list contains 2,080 entities. We randomly pick 57 out of them as the seed set. Some examples are shown in Table 1. After the Candidates Generation phase (Section 3.1), 62,708 candidates are extracted from our corpus. We consider all candidates with capitalized ratio $\frac{|C_t|}{|C_t|+|U_t|} < 0.1$ as negative seeds. Our object is to rank all candidates and make sure the 2,080 entities can rank higher with the 57 positive seeds and auto-generated negative seeds.

Microsoft Web N-gram services are used to build Semantic Pattern Graph. It is a cloud-based platform which extract N-grams from web-scale data of Bing. Since we only consider patterns within window size one, only the bi-gram portion of Web N-gram data is used in our experiment. We also filter out bi-grams with small relative counts such that arbitrary combinations would not contributes to the constructed **SPG**. For the major smoothing parameters, we set $\alpha = 0.3$ and $K = 50$ as default values.

All the large-scale experiments are done in Microsoft Cosmos platform.

office	windows xp	visual studio	sql server
office 365	silverlight	sharepoint 2013	windows 8
windows 7	biztalk	active directory	asp
azure	outlook	windows 2003	outlook

Table 1: Project/Product Seeds from Microsoft

5.2 Baselines and Evaluations

To show that our pipeline performs best for enterprise entity extraction, we compare our work with several baselines as follows. The comparison with existing entity extractors are not shown here due to 1) some of them are not public (e.g., Knowitall and NELL) and they rely mainly on Hearst pattern or high-precision/low-recall patterns, which leads to low coverage in enterprise setting 2) some of them are slow and performance on a small portion is not comparable (e.g., Spied from Stanford).

1. **Count:** Measure the ranking of the term solely by its count of capitalized form in the whole corpus, *score* = log(*count* + 1).
2. **Capi:** Measure the ranking of the term by Capitalized

Method	P@50	P@100	P@200	P@300	P@500
Count	0.38	0.36	0.385	0.383	0.37
Capi	0.16	0.16	0.165	0.15	0.138
Hybrid	0.46	0.61	0.615	0.613	0.59
NB	0.74	0.71	0.72	0.687	0.66
SmooNB	**0.88**	0.81	0.82	0.76	0.744
SemiNB	0.84	0.77	0.78	0.753	0.712
Our System	0.86	**0.85**	**0.84**	**0.843**	**0.824**

Table 2: Precision at top K results for project/product extraction

Ratio: $score = \frac{|C_t|}{|C_t|+|U_t|}$, where C_t and U_t denote the capitalized and uncapitalized mentions in the corpus.

3. **Hybrid:** Rank terms using both count and capitalized ratio: $score = \log(count + 1) * \frac{|C_t|}{|C_t|+|U_t|}$.

We applied both semi-supervised method and SPG-based smoothing to ease the sparsity problem. To study the effectiveness of both factors, we design several ablations as follows

1. **NB:** Use pure Naive Bayes model as the ranker and classifier to extract entities.
2. **SemiNB:** Use Semi-supervised Naive-Bayes model as the ranker and classier. The smoothing step is skipped.
3. **SmooNB:** Use pure Naive-Bayes as the ranker and classifier, apply smoothing after the weights of patterns are learned.
4. **Our System:** Use Semi-supervised Naive-Bayes model as the ranker and classier, and also apply smoothing after the weights of patterns are learned.

5.3 Extracting Projects/Products

In this section, we apply our framework on Microsoft Q/A Forum data and show the main result of projects/products extraction. We also conduct parameter study for Semantic Pattern Graph smoothing afterwards.

5.3.1 Main Results

Since we model Enterprise Entity Extraction problem in a ranking framework, several ranking-based evaluation measures are proposed to assess both precision and recall of the result. Two intuitions are emphasized for the result we yield: 1) The 2,080 entities should rank high on the whole, 2) The top results are real entities with high probability. These intuitions align with the natural of Enterprise Knowledge Base, that is with very high precision first being guaranteed, we then consider about coverage. Using this philosophy, we design three sets of evaluation measures as follows: a) Precision, evaluate the precision of top results from Prec@50 to Prec@500. b) Average Precision: AP@500, AP@1000, AP@2000 as

$$AP@n = \frac{\sum_{k=1}^{n} P(k)}{N}$$

where $P(k)$ means the precision at cut-off k in the entity list, N means the number of relevant entities in total, and c) Discounted Cumulative Gain: DCG@500, DCG@1000 and DCG@2000 as

$$DCG@n = \sum_{k=1}^{n} \frac{\mathbb{1}(e_k \in \mathbb{E})}{\log_2(k+1)}$$

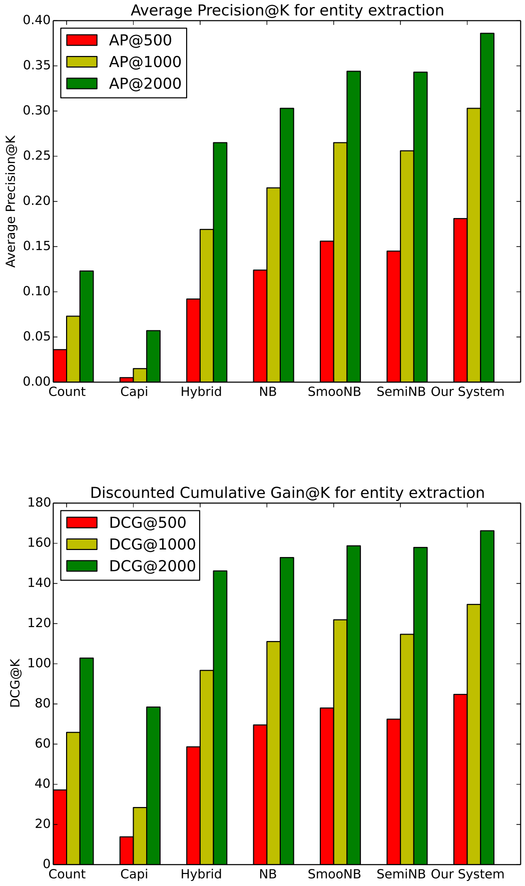


Figure 7: Average Precision and Discounted Cumulative Gain comparison

Where $\mathbb{1}(e_k \in \mathbb{E})$ is the indicator function showing if k-th item is an enterprise entity. Precision measurement emphasizes the top entity quality and AP/DCG measure the balance of both precision and recall. The result is shown in Table 2 and Figure 7.

From the result of Table 2, we observe that our system achieves above 80% precision even for top 500 results and the two baselines **Hybrid** and **NB** only achieves 59% and 66% respectively on top 500. To evaluate the Semi-supervised mechanism and Semantic Pattern Graph smoothing separately, we observe that both **SmooNB** and **SemiNB** can alleviate sparsity and enhance precision. It aligns with our intuition that Semi-supervised Learning can collect more patterns to capture entities with few mentions, while SPG leverage public signals to assign more accurate scores to each pattern. Table 2 also tells us that only applying smoothing can have better performance than only using semi-supervised learning. However, combing these two methods helps us to collect more low-precision patterns and assign them high confident scores, thus can yield best performance.

Figure 7 shows the performance on AP and DCG, with similar conclusion can be drew. AP and DCG are widely used to evaluate ranking qualities such as web search engines. We observe that using either Smoothing or Semilearning outperforms the local ranker using only signals from corpus and positive seeds. Similar to precision, SPG smoothing improves more in terms of AP and DCG as well.

5.3.2 *Parameter Study*

In this section, we study the two major parameters used in our smoothing framework: 1) Smoothing weight α, 2) Top pattern size K. Changing α studies the balance between local signals from enterprise corpus and public signals from the web, while changing K studies the effective size of trustworthy patterns using only local signals.

We plot the 3-dimensional chart in Figure 8, in which we control α from 0 to 1 with size-0.03 steps and K from 10 to 200 with size-10 steps respectively, to evaluate precision of entities on top 500 results. We first observe that the performance goes up after we gradually apply smoothing to the patterns, and it reaches the best precision of 83.05% at $alpha = 0.32$ and $K = 50$. If we observe α curve standalone, we see that precision is positively correlated to α before it hits 0.32. After that, it keeps relatively high precision if $\alpha < 0.7$ and drops drastically if $\alpha > 0.7$. It makes sense in the sense that 1) Properly leveraged public signals can effectively alleviate sparsity and 2) Public signal itself cannot capture the unique semantics used in enterprise.

On the other hand, if we observe K curve standalone, we find the similar situation where the peak performance is achieved when K is reasonably large but not too large, say 50. As discussed before, our philosophy of picking K is to cover enough "high-precision/low-recall" patterns. If K is small, the top patterns can not capture most aspects of extracted class C, and therefore, smoothing phase may lead to strongly biased adjustment. As we can see from the figure 8, performance when $K = 10, \alpha = 0.32$ is even lower than the original one. If K is too large, due to sparsity issues, many $low - precision$ patterns may also be used as smoothing seeds and contaminate the rest of the Pattern Graph.

Currently we use intuitions to choose $K = 2\%$ of all patterns and $\alpha = 0.3$. More study on this issue will be covered in our future work.

5.4 Pattern Study

In this section, we conduct experiments to look closer to patterns and answer two questions: 1) Why previous entity extractors with high-precision/low-recall patterns will fail in enterprise setting? 2) How well the Semantic Pattern Graph performs to ease sparsity in a real case?

5.4.1 *Pattern Coverage Study*

To answer the first question, we implemented several pattern extractors used in previous systems: 1) NELL, 2) Knowitall and 3) SPIED. As discussed previously, NELL and SPIED used high-precision/low-recall patterns with part-of-speech (POS) restrictions to identify entities, while Knowitall uses mainly Hearst patterns and predefined templates for entity extraction. For Knowitall, we included all pattern templates mentioned in their paper [9]. For NELL, we strictly follow their Coupled Pattern Learner [6] approach, where the patterns follows its POS rules and at most 100 instances/entities and 5 patterns are promoted. Patterns are ranked by precision and instances are filtered out unless the number of times it co-occurs with promoted pattern is at least three times more than the number of times it co-occurs with patterns left. For Spied, since the extracting details are not disclosed in the paper, we use the same POS restrictions that NELL used to extract patterns.

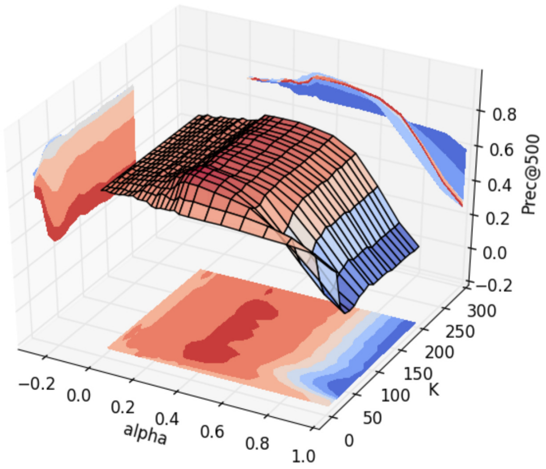


Figure 8: Parameter Study for smoothing weight α and top pattern size K. The performance is evaluated based on Precision@500

System	Knowitall	NELL	Spied	Our System
Coverage (%)	21.5	54.4	54.3	**81.7**

Table 3: Coverage of projects/products using extracted lexical patterns

Table 3 shows the coverage of the 2,080 labeled projects using different lexical pattern set generated by four systems. We observe that our system can potentially extract more than 80% of target enterprise entities, while in the meantime, NELL/Spied/Knowitall can only find less than 55% entities. That is to say, in a sparse setting, even the best existing classifier/ranker would lose about half of the entities. Knowitall has the lowest coverage due to the limited pattern templates it uses. Hearst patterns have poor coverage in enterprise corpus. NELL and Spied use more general patterns confined by specific POS templates can yield better entity coverage than Knowitall, but are still deficient for a sparse setting. Because the POS constraints normally require verbs in the pattern or require explicit relations to another noun phrases. However, many enterprise entities lack of the redundancy of such patterns. For example in our system, two patterns "+platform" and "−native" are good signals for identifying enterprise entities, but will not be considered by precious web-scale extractors. Note that 20% of the entities in Microsoft are still unreachable by our patterns, some of them are never mentioned in the corpus and some of them are mentioned only a few times with non-informative patterns (the general patterns which are not exclusive to projects/products).

A reasonable concern of introducing less restricted patterns is that it potentially lower our confidence on judging every single pattern. Hence we introduced smoothing based on Semantic Pattern Graph to get accurate evaluation of each pattern.

Pattern	Score Before	Smoothing Factor	Score After	Mention Count	Examples
+debugger	0.157	0.895	0.378	136	... from an attached **VS10** *debugger.* ...triggered the **jit** *debugger* dialog...
+backend	0.647	0.983	0.748	108	Both the **SQL Cluster** *Backend* and... ...instance of **FAST Search Server** *backend*...
+verification	-0.094	0.704	0.145	77	Failed **authenticode** *verification* of payload.
-existing	0.456	-0.532	0.160	1789	...migrate the *existing* **ORANGE** connection.. ..want all *existing* **assignments** such for...
-new	0.253	-0.995	-0.121	8239	...to create *new* **experience** I get... ...in *new* **Software Distribution** policies...

Table 4: Smoothed patterns along with the score before smoothing, smoothing factor and score after smoothing, and the example entities they appear with. Scores in columns 2-4 are normalized to [-1, 1], where positive score means it is good pattern for extracting product/project, and vice versa. Score After is the linear combination of Score Before and Smoothing Factor using $\alpha = 0.3$. Mention Count indicates how popular the pattern is in the corpus. Entity examples for the first three patterns are good enterprise entities which are promoted due to the promotion of their patterns, and the entities for last two patterns are the mistakenly classified entities which are demoted due to the demotion of their patterns.

5.4.2 SPG Smoothing evaluation

As discussed in previous sections, the SPG smoothing improves the pattern accuracy in two ways: 1) Demote Type II - Bad General Patterns. 2) Promote Type III - Sparse Good Patterns. Table 4 revisits the examples we have analyzed and shows the smoothing result for them and entities they extracted. '+debugger', '+backend' and '+verification' are Type III patterns, which are good indicators for projects/products but are underrated in our corpus. Some good patterns, like '+verification', are even evaluated as negative before smoothing. Since all these three patterns have clear **Belonging** relation with top patterns like '−develop' (they have highly correlated semantics to '−develop', but much more specific), their smoothing factor is usually pretty large. In the chart, we can see their scores after smoothing is promoted a lot and the entities appear with them can higher rank. On the other hand, '−existing' and '−new', as we analyzed, are Type II patterns with strongly biased signals. They don't indicate any semantics specifically correlated to project/product. However, they have large positive scores before smoothing due to the fact that they are commonly used with any positive seeds. For those general patterns, the top patterns, like '−develop', have clear **Belonging** relations with them. Therefore, their smoothing factor is pretty negative and the pattern is demoted after smoothing.

One more thing we need to notice is that, Type II patterns are just too general, the ideal case is that they should play no role in identifying enterprise entity nor non-enterprise entity. Therefore, the demoted score should be close to 0. And our method may potentially over-penalize general patterns if they happen to not appear with seed entities, with low probability (Positive seeds are usually very common entities in the corpus).

According to the mention count column, we can see that Type II patterns are much more common than Type III patterns when they appear with our enterprise entities. It complies with our intuition that 1) Type II patterns are too general that they may happen to appear with many positive seeds and 2) Type III patterns are too specific that positive seeds never appear with them. These examples, along with the performance comparison in Section 5.3, we conclude that Semantic Pattern Graph smoothing works well for resolving sparsity in enterprise setting.

6. CONCLUSION AND FUTURE WORK

This paper introduced our framework for enterprise entity extraction, which involves semi-supervised bootstrapping entity extraction framework using seed sets. It is the first work to introduce the concept of Semantic Pattern Graph into entity extraction problem and model the semantics between contextual patterns to enhance entity coverage and precision. With detailed analysis, we show that in a sparse setting, leveraging public web-scale data to understand semantic relations between lexical patterns can effectively alleviate the sparsity. Hence, many entities with very little evidence can be possibly correctly extracted. In the experiment part, we also show that the smoothing algorithm using SPG can significantly improve precision and recall of learned enterprise entities. We also conduct several other experiments to show our smoothed extractor outperforms methods without understanding pattern relations. Moreover, the Semantic Pattern Graph and its smoothing algorithm is general enough to be applied into many other Information Extraction tasks on closed-domain corpus (even other NLP tasks).

In future work, we plan to extend our size-1 patterns into more general patterns to adapt more complicated environment. Another research direction is to utilize Semantic Pattern Graph in a more complicated way. Conducting clustering on lexical patterns in a SPG may help us find several pattern groups with similar semantics, such that the way of picking seed patterns can be more balanced in different pattern groups. We also plan to generalize the Semantic Pattern Graph component into many other IE tasks, including relation extraction and entity typing.

7. ACKNOWLEDGEMENTS

Research was sponsored in part by the U.S. Army Research Lab. under Cooperative Agreement No. W911NF-09-2-0053 (NSCTA), the Army Research Office under Cooperative Agreement No. W911NF-13-1-0193, National Science Foundation IIS-1017362, IIS-1320617, and IIS-1354329, HDTRA1-10-1-0120, NIH Big Data to Knowledge (BD2K) (U54), and MIAS, a DHS-IDS Center for Multimodal Information Access and Synthesis at UIUC.

8. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *ACM DL*, pages 85–94, 2000.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *Dbpedia: A nucleus for a web of open data*. Springer, 2007.

[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[4] S. Brin. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, pages 172–183. Springer, 1999.

[5] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.

[6] A. Carlson, J. Betteridge, R. C. Wang, E. R. Hruschka Jr, and T. M. Mitchell. Coupled semi-supervised learning for information extraction. In *WSDM*, pages 101–110, 2010.

[7] W. W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *AAAI*, pages 335–342, 1999.

[8] D. Downey, O. Etzioni, S. Soderland, and D. S. Weld. Learning text patterns for web information extraction and assessment. In *AAAI*, pages 50–55, 2004.

[9] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall:(preliminary results). In *WWW*, pages 100–110, 2004.

[10] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Methods for domain-independent information extraction from the web: An experimental comparison. In *AAAI*, pages 391–398, 2004.

[11] O. Etzioni, A. Fader, J. Christensen, S. Soderland, and M. Mausam. Open information extraction: The second generation. In *IJCAI*, pages 3–10, 2011.

[12] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *EMNLP*, pages 1535–1545, 2011.

[13] D. Freitag. Toward general-purpose learning for information extraction. In *ACL*, pages 404–408, 1998.

[14] S. Gupta and C. D. Manning. Improved pattern learning for bootstrapped entity extraction. *CoNLL-2014*, page 98, 2014.

[15] S. Gupta and C. D. Manning. Spied: Stanford pattern-based information extraction and diagnostics. *Sponsor: Idibon*, page 38, 2014.

[16] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Conference on Computational Linguistics*, pages 539–545, 1992.

[17] A. Lenci. Distributional semantics in linguistic and cognitive research. *Italian journal of linguistics*, 20(1):1–31, 2008.

[18] H. Poon and P. Domingos. Unsupervised ontology induction from text. In *ACL*, pages 296–305, 2010.

[19] E. Riloff. Automatically generating extraction patterns from untagged text. In *AAAI*, pages 1044–1049, 1996.

[20] E. Riloff, R. Jones, et al. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479, 1999.

[21] M. Schmitz, R. Bart, S. Soderland, O. Etzioni, et al. Open language learning for information extraction. In *EMNLP-CoNLL*, pages 523–534, 2012.

[22] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[23] C. Sutton and A. McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, pages 93–128, 2006.

[24] M. Thelen and E. Riloff. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *ACL*, pages 214–221, 2002.

[25] R. C. Wang and W. W. Cohen. Language-independent set expansion of named entities using the web. In *ICDM*, pages 342–350, 2007.

[26] R. C. Wang and W. W. Cohen. Iterative set expansion of named entities using the web. In *ICDM*, pages 1091–1096, 2008.

[27] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *ACL*, pages 189–196, 1995.